International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 10 Issue: 12
DOI: https://doi.org/10.17762/ijritcc.v10i12.5896
***Article Received**: 10 October 2022 **Revised**: 14 November 2022 **Accepted**: 16 December 2022*

FPGA Implementation of Double Precision Floating Point Multiplier

Mohd Abdullah[1], Dr. Bharti Chourasia[2]
[1]PhD Scholar, Department of Electronics & Communication Engineering,
SRK University, Bhopal, India
mab434@gmail.com
[2]Associate Professor & HOD, Department of Electronics & Communication Engineering,
SRK University, Bhopal, India
bharti.chourasia27@gmail.com

Abstract—High speed computation is the need of today's generation of Processors. To accomplish this major task, many functions are implemented inside the hardware of the processor rather than having software computing the same task. Majority of the operations which the processor executes are Arithmetic operations which are widely used in many applications that require heavy mathematical operations such as scientific calculations, image and signal processing. Especially in the field of signal processing, multiplication division operation is widely used in many applications. The major issue with these operations in hardware is that much iteration is required which results in slow operation while fast algorithms require complex computations within each cycle. The result of a Division operation results in a either in Quotient and Remainder or a Floating point number which is the major reason to make it more complex than Multiplication operation.

Keywords- Floating Point Arithmetic, Multipliers, Digital Arithmetic, FPGA, DSP48E.

I. INTRODUCTION

Floating-point matrix computation is widely employed in the disciplines of image processing, deep learning system control, and digital signal processing. Its computational efficiency does have a direct influence on the overall performance of the system. Various systems for accelerating matrix calculations have arisen in recent times, such as FPGA CPU, GPGPU, and software libraries. (FPGAs) are perfect for speeding matrix calculations as a co-processing basis. FPGAs surpass general-purpose CPUs and GPGPU system in terms of the long productivity, according to several studies. Its fully programmable and substantial logic resources, notably the huge number of embedded DSPs and BRAMs (Block RAMs), lay the foundation for improved matrix computing performance. The fundamental purpose of many algorithms and associated hardware designs is to create a balance between resource demands and performance. Methods for multiplication of fixed-point matrices, for example, have been presented. The designers created a full FPGA coprocessor for multiplications. Similar systolic array architectures were presented due of the large data flow and processing speed. The multiplications structure has also been studied by several scholars. Matrix computation still has a number of issues, despite all of the research that has been done. Older systems had limited processing power and could only handle fixed-point information and small or medium-sized matrix. We intend to create a matrix computation acceleration (MCA) unit capable of processing large matrices with high data accuracy. Second, dealing with matrix equations of varying sizes is difficult when employing a fixed structure. As a result, assessing the structure's flexibility is critical and helpful. Furthermore, most previous work can only do a single matrix operation. A specialized matrix operation, on the other hand, is typically sufficient in many technological applications nowadays. It needs a large variety of matrix processes, and also the ability to perform a series of matrix operations with the same structure.

Now for about two decades, (Field Programmable Gate Arrays) have been around. Since then, they've increased in popularity and have become a standard way to build digital circuits. Because of advancements in processing technologies, the logic capacity of FPGAs has significantly risen, giving them a viable implementation choice for bigger and more complex designs. Additionally, the fully programmable of their logic and routing resources has a massive effect on the finished device's space, speed, and energy consumption. Because of its programming and routing interfaces, FPGAs are much more versatile more general-purpose that standard cell ASICs, but they are also bigger, and consume more energy. Improvements in processing technologies, on the other hand, have necessitated and enabled a number of alterations to the fundamental FPGA structure. These developments are aimed at increasing the overall efficiency of FPGAs in order to close the gap between them and ASICs. Programmable Logic Arrays (FPGAs) are pre-fabricated silicone devices that may be electronically programmable to create practically any form of digital schematic in the field. FPGAs provide such a cheaper option & speed to market for low and medium volume manufacturing

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 10 Issue: 12
DOI: https://doi.org/10.17762/ijritcc.v10i12.5896
***Article Received**: 10 October 2022 **Revised**: 14 November 2022 **Accepted**: 16 December 2022*

than Embedded Integrated Devices (ASICs), which frequently require a lot of resources of time and cash to get the first device. FPGAs, on either hand, may be set up in less than a moment and cost anywhere between a few hundreds to a few thousands of dollars.

II. Floating-Point Unit (FPU)

Wherever the computing unit's floating-point unit is an IEEE-754 compliant integrated unit with double resolution. Fixed to float conversions, float to float converting, floating-point addition (64-bit), and floating-point multiplication is among the four floating-point processes it can execute (64-bit). FPU is given a 2-bit operation code to choose which operations can be performed. The enabled signal instructs the FPU to begin performing the needed operation.

In the Xilinx-ISE tool, IP cores are provided to solve these essential floating-point calculations. However, because our computing unit is aimed towards ASIC creation, IP cores are not an option. To provide a comprehensive hardware descriptions of these processes, all four floating-point processes are implemented using an algorithmic state machine (ASM) in Verilog HDL.

Each floating-point operation that the FPU performs requires many cycles to accomplish. As a result, a handshaking mechanism between the control module and the FPU is devised so that the control module can track the FPU busy state through using FPU ready signal. The floating-point multiplying is also done using a fixed-point multiplication.

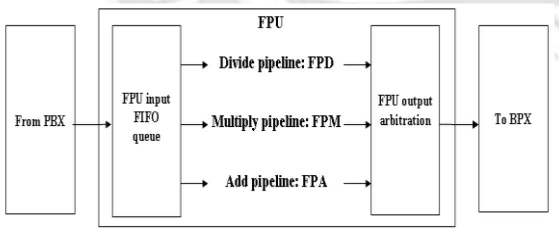

Figure: 1.1: Floating-Point Units

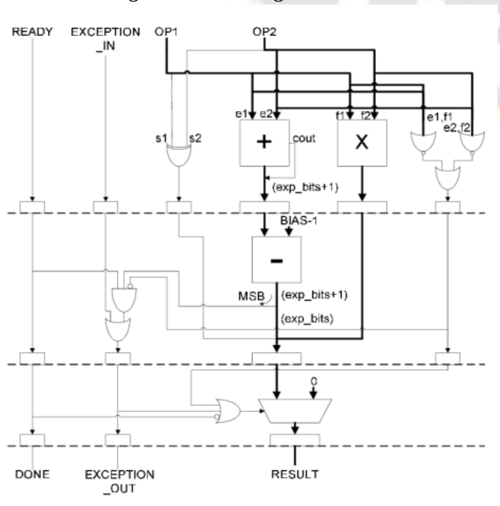

Figure 1.2: Double-precision Floating-point Multiplication

ASM is used to implement the floating-point added technique, which is a step-by-step operation. Figure 1.1 depicts the whole technique for adding two double-precision floating-point values, such as ADBL and BDBL. There are several phases involved in floating-point multiplying, as shown in Figure 1.2. Figure 1.1 shows the procedures involved in converting a fixed-point value to its corresponding floating-point number and depicts the whole technique for converting a floating-point value to a set amount. The data route of the computing unit is utilized with the floating-point unit described in this article to conduct the essential operations for polynomial solution. The following part describes the design and implementation of the data route and its peripherals.

III. Related Work

Machupalli Lahari et.al. (2020) “Efficient Floating-Point HUB Adder For FPGA” An effective floating-point HUB adder for FPGA was designed and built. Both a single route and a double route approach are used to build and study the HUB adder. When compared to the single path technique, the power and latency of the double path strategy are bearable. Furthermore, spurious power suppression is described as a way for reducing Dynamic power consumption. The propose doubled path HUB floating point adder with SPST decreases power and latency by 15% and 13%, respectfully, when compared to the current double path HUB precision floating - point adder.

D S Bormane et.al. (2020) “Acceleration Techniques using Reconfigurable Hardware for Implementation of Floating Point Multiplier” Two Algorithms for doing a 24*24 significant multiplying for IEEE single precision values are described. When it comes to LUTs, the first technique takes up less space than the second method. Furthermore, the computing time needed to do the multiplying is around one-third of that required by the second method. The suggested multiplier surpasses all prior techniques, and the power delay parameter has been shown to be quite useful.

Alahari Radhika et.al. (2020) “Low Complexity Fused Floating Point FFT Using CSD Arithmetic for OMP CS System” The application-driven hardware fusing and Canonical sign digit-driven shift accumulation-based mantissa calculation approaches are used to minimize the computational complexity that occurs in floating point arithmetic-based complex FFT computing. Hardware sharing is accomplished by utilizing the FFT butterfly structure's inherent redundant computing features, which significantly minimises computational complexity overhead. It is also shown that during FPU multiplication, utilizing a multiplier-less mantissa calculation reduces complexity significantly. Finally, the metrics of these two numerical optimization are evaluated in a twiddle factors optimized FFT structure. The basic link among floating point

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 10 Issue: 12
DOI: https://doi.org/10.17762/ijritcc.v10i12.5896
***Article Received**: 10 October 2022 **Revised**: 14 November 2022 **Accepted**: 16 December 2022*

unit and performance measures of both hardware and twiddle factors optimization approaches has been carefully investigated, and FPGA hardware creation for qualitative and quantitative testing has been undertaken.

V. Ramya et.al. (2020) "Low power single precision BCD floating–point Vedic multiplier" A low-power, delay-efficient BCD-floating point multiplication (BCD-FPM) for single precision is developed using the UT sutra. Methods I and Method II have been proposed for BCD-FPM, with BFPM being produced applying KSA to compare the findings. According to the statistics, the BCD-FPM Method II exceeds the BFPM by 73.41 percentage in terms of strength and 30.37 percent in terms of delay, while the BCD-FPM Methods I out performs the BFPM by 59.48 percentage in terms of strength and 6.9 percent in terms of latency. To boost performance even more, BCD-FPM Method II is parallelized. When comparing to design without pipes, the power-delay result of innovation by 43.44 percent as the size of a pipelined structure grows. As a consequence, the pipelined BCD-FPM System II is outperformed both the BCD-FPM Method I and the BFPM Methods.

Mohamed Al-Ashrafy et.al. [2019] "An Efficient Implementation of Floating Point Multiplier" For the Virtex-5 FPGA from Xilinx, the proposal outlines a quick implementation of an IEEE 754 decimal numbers floating point multiplication. VHDL is used to create a technology-independent pipelined design. The multiplier implementation handles overflow and underflow circumstances. Rounding is not employed when using the multipliers in a Multiple and Accumulation (MAC) unit to improve accuracy. With a three-clock-cycle delayed, the design provides 301 MFLOPs. The multiplier was compared to a Xilinx floating point multiplication core.

Gokul Govindu et al.[2019] "Analysis of High-performance Floating-point Arithmetic on FPGAs" Inside the high-performance and experimental computational fields, the suggested FPGAs are rapidly being used to develop floating-point based hardware accelerators. The floating-point multiplication and adder/subtractor units are investigated in this study using the numbers of pipeline stages as a parameter and throughput/area as a metric. The authors achieve throughput ratios of much more than 240Mhz (200Mhz) for singles (double) precision processes by substantially pipelining the devices. To demonstrate the influence of floating-point modules on kernels, the author develops a multiplications kernel based on our floating-point units & show that state-of-the-art FPGA devices can reach roughly 15GFLOPS for single (double) precision floating-point based multiplications (8GFLOPS). The authors also show that FPGAs may outperform general-purpose CPUs by up to 6 times in terms of GFLOPS/W (performance per unit power) (for single precision). The authors then go into the effects of floating-point unit on the construction of energy-efficient matrices multiple kernels structure.

Soner Yes et.al. (2018) "Experimental Analysis and FPGA Implementation of the Real Valued Time Delay Neural Network Based Digital Predistortion" RVTDNN-based Digital Predistortion has indeed been experimentally investigated on hardware, and an FPGA implementation with really resource -constrained use has been given. Effective usage of the DSP48 basic blocks' dedicated instruction register, restricted learning, and contribute to creating linearly approximation artificial neurons are the underlying drivers of the difference in resource consumption and operating clock frequency. The proposed approaches allow for compact and adaptable FPGA implementations of complex neural networks with such a group of neurons and increased pulse bandwidths.

Junzhong Shen et.al. (2018) "Towards a Multi-array Architecture for Accelerating Large-scale Matrix Multiplication on FPGAs" The focuses are on the architectural expansion of the linear array structure for matrix multiplication on FPGA by providing a highly customizable and scalable multi-array architecture. We employ a work-stealing strategy to establish workload balancing among PE arrays. An effective analytical model is constructed to find the optimum design possibilities for the architectural expansion. According to testing data, our ideal extension of the linear array design may deliver the best performance and processing efficiency.

Y. R. Annie Bessant et.al. (2018) "Analysis of Area and Delay for Floating Point Matrix Multiplication" The different floating point multiplications designs, each having a memory and velocity trade-off, are demonstrated. Matrices are the most essential part in terms of the size & delayed consumed for multiplications. The recommended design employs floating point multiplication and UrdhvaTiryagbhyam multipliers. According the findings of the performance monitoring, our method's performance improved in terms of delay and area. With a design frequency of 189.517 MHZ on the virtex-6 xc6vlx240t FPGA, the algorithms can retain top performance.

IV. PROPOSED METHDOLOGY

To obtain the final outcome of Multiplying, a summing of these partial products is calculated using ADDER. The number of partial products created grows in proportion to the bit size of the Multiplication. The 256 partial products are created for a 16 bit adder. Booth's Encode, a method for reducing the amount of partial products formed during multiplications, can help lower this quantity.

Components and connectivity can be shared across several data routes using data path merging techniques, resulting in a shared data path. When space conservation is a top issue, such sharing is extremely vital. Resource sharing has always been

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 10 Issue: 12
DOI: https://doi.org/10.17762/ijritcc.v10i12.5896
***Article Received**: 10 October 2022 **Revised**: 14 November 2022 **Accepted**: 16 December 2022*

limited to components and bit widths of the same size. While this saves space, it does not allow for the full utilization of available shares. Because floating-point data routes are complicated and comprise many components with various bit widths, this is a significant hurdle to merging them.

The exponent and mantissa components of a floating-point number, for example, are computed using floating-point arithmetic data pathways. The exponent and mantissa components for single-precision would typically be 8-bits and 24-bits, respectively. Exponent and mantissa constituents for double-precision would typically be 11-bits and 53-bits, respectively. 8-bit, 24-bit, 11-bit, 53-bit, and 32-bit component will be used in converting operations between integer, single-precision, and double-precision formats. This results in a large number of various bit-width components that might be shared, and limiting sharing to only similarly sized components would significantly limit the possibility for space savings.

Sharing components with varied bit-widths is required to maximise resource sharing. Whenever the data paths of two distinct operations, also with a 24-bit adder and the other with a 32-bit adder, are combined, the two adders should indeed be replace with a single shared 32-bit adder, as illustrated in Figure 1.3.

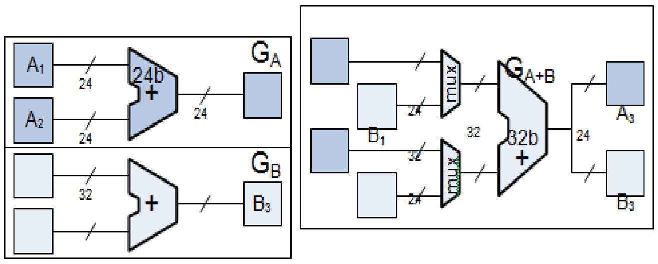

Figure 1.3: Merging of 24-bit and 32-bit adders

When a bigger component or interconnection is merged, the bit-alignment problem is solved by determining how a smaller component or interconnection should be aligned inside the larger component or connection. Some components require precise alignment to work effectively, whereas others can function successfully even if they are not aligned. When a chain of subsequent components and interconnects is maintained and must be aligned precisely in relation to one another, the problem becomes much more problematic.

The data route merging approach allows resources available with mismatched bit widths to be combined, but it does not take into consideration alignment difficulties. Furthermore, there has been relatively little research on the bit-alignment problem in the literature. Many works presume a constant bit width throughout one design, or neglect the bit representation of signals entirely, or require the user to manually adjust the size of the data words to matching the hardware size. Bit-alignment errors, on the other hand, had to be manually addressed, which was a time-consuming procedure. Because a pair of connections that are recognized as shared may turns out to be non-shareable owing to alignment concerns, ignoring bit-alignment during data route merges may result in sub-optimal merger outcomes. As a result, a solution to this situation is urgently required. A unique technique was created to satisfy the demand for a bit-alignment method during data plane merging. The introduction of this unique solution is the major topic of this chapter. This new bit-alignment approach permits the merging of resource with non-matching bit-widths while addressing the bit-alignment problem, optimizing area savings and allowing for more sophisticated data path combining.

Another drawback of the custom FPU generation process described in Chapter 4 is that it ignores the volume and throughput trade-off among hardware implementation and software emulation instruction. A selection of the floating-point instructions required for the applications is implemented in the hardware FPU, and the floating-point operations that are not implemented in the hardware FPU are emulated in software when developing a bespoke FPU. As a result, there are various trade-offs to consider as floating-point instructions are off-loaded onto specialised floating-point hardware. In general, the more processes done in hardware, the more space is utilised, but the fewer cycles are required to finish the application's execution. More hardware, on the other hand, may cause the clock time to lengthen. This is especially the case in datapath merging, where adding multiplications to the crucial path might cause delays.

As a result, a quick design space exploration was carried out in this chapter to study these trade-offs in the FPU generation methodology.

Starting with a fixed alignments component, the consistency chart is traversed to promote the needed alignments all through the graph. An alignments tree is used to hold the traversal route and alignments data. In the compatibility graph, this is repeated for each fixed alignments components. After that, the alignments trees are integrated into a single alignments tree that is used to trim and annotation the compatibil graph. The maximum weight coterie may then be solved using the processed compatibility graph, as shown in. Data structures defining components and interactions are created from the datapath descriptions. Using a library of VHDL descriptors for each components, the components data structures are then utilised to construct VHDL constituent declarations and port mapping. All of the functional departments that were developed during the creation of the floating-point datapaths in VHDL were separated and grouped to create the library of VHDL description. To make signal assignment easier, each component's input and outputs ports are given a distinct signaling name in the port mappings.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 10 Issue: 12
DOI: https://doi.org/10.17762/ijritcc.v10i12.5896
***Article Received**: 10 October 2022 **Revised**: 14 November 2022 **Accepted**: 16 December 2022*

The signal assignments are generated using the connecting data structures. The auxiliary information instructs the VHDL generation where processing elements are inserted and how to construct the datapath to complete each task.

V. RESULTS

The graphs depict the runtime as a function of the area. Each of the combinations that gave the quickest runtimes is shown by a circle. The higher the area and the shorter the runtime, as predicted, the more operations executed in hardware. Over a certain point, the majority of the graphs show declining re-turns. This because when more data pathways are integrated into the FPU, the clock period increase, and the amount of cycles saving by utilising hardware (rather than software emulation) decreases as lower-ranked operations are performed in hardware. With the exception of epic and mpeg2enc, the runtime gradually increases in all scenarios. The low-ranked commands were implemented so seldom in these circumstances that the cycles saved by employing hardware were insufficient to compensate for the increased delay.

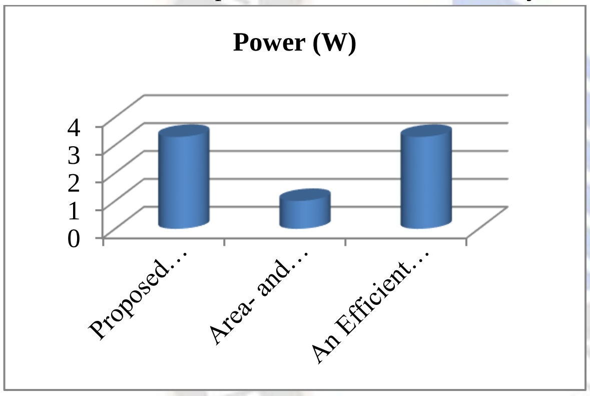

Figure 1.4: Power Analysis-I

Figure 1.4 is showing power consumption of the proposed work and the previous author's work. The variety of Possible Floating Point Multipliers in the portion with Power (W) in diagram is 3.294. Iterative single/double-precision merging floating-point/multiplier on FPGA (2015) has an area and power efficiency of 1. And the answer is 3.294 in An Actual Integration of Floating Point/Multiplier.

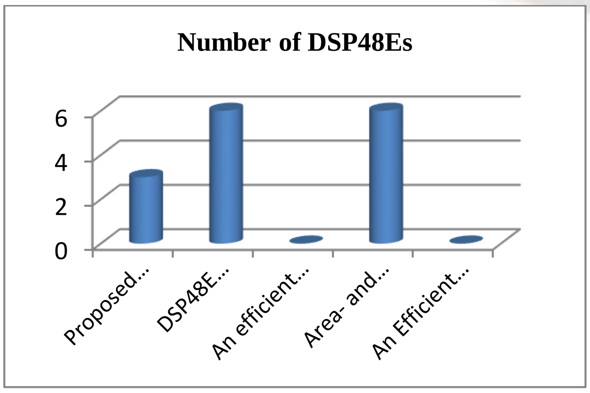

Figure 1.5: Number of DSP48Es

Figure 1.5 shows the number of DSP48E used in the previous author's work and proposed work.

VI. CONCLUSION

The implementation is described in this dissertation. In a suggested technique, a process for automatically producing FPUs tailored at the instruction level was provided, which included datapath merging to reduce area. The merging of complicated floating-point datapaths was approached using a maximum weight clique technique. Customizable FPUs were created for several Mediabench application and compare against a reference FPU that supported all floating-point operations. Minimizing the floating-point instruction set resulted in significant space savings, while datapath merging resulted in even more savings. The FPU size was decreased by an average of 51% by reducing the floating-point instruction set to the bare minimum necessary for the application. The FPU area was lowered by an average of 65% by conducting instruction reduction and then merging the floating-point datapaths. The results demonstrated the efficacy of modifying FPUs at the instruction level, as well as the efficacy of datapath merging as a way of reducing area.

REFERENCES

[1] Raul Murillo , Alberto A. Del Barrio, Guillermo Botella , Min Soo Kim, HyunJin Kim and Nader Bagherzadeh “PLAM: a Posit Logarithm-Approximate Multiplier for Power Efficient Posit-based DNNs”2021.

[2] Geetam Singh Tomar, Marcus Llyode George, Abhineet Singh Tomar “Multi-precision binary multiplier architecture for multi-precision floating-point multiplication” 2021.

[3] Hamzah Abdel-Aziz ,Ali Shafiee ,Jong Hoon Shin , Ardavan Pedram , Joseph H. Hassoun “Rethinking Floating Point Overheads For Mixed Precision Dnn Accelerators” 2021.

[4] Varun Gohil, Sumit Walia, Joycee Mekie, Manu Awasthi “A Floating-Point Representation for Error-Resilient Applications” 2021.

[5] N. Bhavani Sudha, Gamini Sridevi “An Efficient Design Of Multiplier And Adder In Quantum-Dot Cellular Automata Technology Using Majority Logic” 2021.

[6] Y Mounica , K Naresh Kumar , Sreehari Veeramachaneni , Noor Mahammad “Energy efficient signed and unsigned radix 16 booth multiplier design” 2021.

[7] Yuheng Yang, Qing Yuan1, And Jian Liu “An Architecture of Area-Effective High Radix Floating-Point Divider With Low-Power Consumption” 2021.

[8] J. Jean Jenifer Nesam ,S. Sivanantham “Efficient half-precision floating point multiplier targeting color space conversion” 2020.

[9] TaiYu Cheng , Yukata Masuda , Jun Chen , Jaehoon Yu , Masanori Hashimoto “Logarithm-approximate floating-point multiplier is applicable to power-efficient neural network training” 2020.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 10 Issue: 12
DOI: https://doi.org/10.17762/ijritcc.v10i12.5896
***Article Received**: 10 October 2022 **Revised**: 14 November 2022 **Accepted**: 16 December 2022*

[10] Thiruvenkadam KRISHNAN , Saravanan S2 "Design of Low-Area and High Speed Pipelined Single Precision Floating Point Multiplier" 2020.

[11] Chuangtao Chen, Sen Yang , Weikang Qian, Mohsen Imani , Xunzhao Yin , Cheng Zhuo "Optimally Approximated and Unbiased Floating-Point Multiplier with Runtime Configurability" 2020.

[12] Zhaojun Lu, Md Tanvir Arafin, Gang Qu "RIME: A Scalable and Energy-Efficient Processing-In-Memory Architecture for Floating-Point Operations" 2020.

[13] Machupalli Lahari, Sonali Agrawal "Efficient Floating-Point Hub Adder For Fpga" 2020.

[14] D S Bormane, Sushma Wadar, Avinash Patil, S C Patil "Acceleration Techniques using Reconfigurable Hardware for Implementation of Floating Point Multiplier" 2020.

[15] Alahari Radhika, Kodati Satyaprasad, and Kalitkar Kishan Rao "Low Complexity Fused Floating Point FFT Using CSD Arithmetic for OMP CS System" 2020.

[16] V. Ramyaa, R. Seshasayanan "Low power single precision BCD floating–point Vedic multiplier" 2020.

[17] Manish Kumar Jaiswal, And Hayden K.-H. So "DSP48E Efficient Floating Point Multiplier Architectures on FPGA" [2019].

[18] MohamedAl-Ashrafy,AshrafSalem,WagdyAnis"An Efficient Implementation of Floating Point Multiplier" [2019].

[19] Ling Zhuo and Viktor K. Prasanna "Sparse Matrix-Vector Multiplication on FPGAs" [2019].

[20] Gokul Govindu, Ling Zhuo, Seonil Choi and Viktor Prasanna "Analysis of High-performance Floating-point Arithmetic on FPGAs" [2019].

[21] Soner Yes¸ Cansu S¸ Ali Ozg ¨ ur Yılmaz "Experimental Analysis and FPGA Implementation of the Real Valued Time Delay Neural Network Based Digital Predistortion" [2018].

[22] Aneela Pathan , Tayab D Memon and Sheeraz Memon "A Carry-Look Ahead Adder Based Floating-Point Multiplier for Adaptive Filter Applications" [2018].

[23] Junzhong Shen, Yuran Qiao, You Huang Mei Wen and Chunyuan Zhang "Towards a Multi-array Architecture for Accelerating Large-scale Matrix Multiplication on FPGAs" [2018].

[24] Y. R. Annie Bessant and T. Latha "Analysis of Area and Delay for Floating Point Matrix Multiplication" [2018].

[25] Yixing Li, Zichuan Liu, Kai Xu, Hao Yu, Fengbo Ren "A Gpu-Outperforming Fpga Accelerator Architecture For Binary Convolutional Neural Networks" [2018].

[26] Vladimir Rybalkin, Alessandro Pappalardo "FINN-L: Library Extensions and Design Trade-off Analysis for Variable Precision LSTM Networks on FPGAs" [2018].

[27] Manish Kumar Jaiswal, and Hayden K.-H So "DSP48E Efficient Floating Point Multiplier Architectures on FPGA" [2017].

[28] Martin Langhammer, Bogdan Pasca "Single Precision Natural Logarithm Architecture for Hard Floating-Point and DSP-Enabled FPGAs" [2016].

[29] Prasad Bharade, Yashwant Joshi, Ramchandra Manthalkar "Design and Implementation of FIR Lattice Filter using Floating Point Arithmetic In FPGA" [2016].

[30] Mohammed Dali , Ryan M. Gibson , Abbes Amira, Abderezak Guessoum and Naeem Ramzan "An Efficient MIMO-OFDM Radix-2 Single-Path Delay Feedback FFT Implementation on FPGA" [2015].

[31] E. George Walters "24-Bit Significand Multiplier for FPGA Floating-Point Multiplication" [2015]